埃拉托斯特尼筛法

过程

考虑这样一件事情：对于任意一个大于1的正整数n，那么它的x倍就是合数（x > 1）。利用这个结论，我们可以避免很多次不必要的检测。

如果我们从小到大考虑每个数，然后同时把当前这个数的所有（比自己大的）倍数记为合数，那么运行结束的时候没有被标记的数就是素数了。

```
// C++ Version
int Eratosthenes(int n) {
  int p = 0;
  for (int i = 0; i <= n; ++i) is_prime[i] = 1;
  is_prime[0] = is_prime[1] = 0;
  for (int i = 2; i <= n; ++i) {
    if (is_prime[i]) {
      prime[p++] = i;  // prime[p]是i,后置自增运算代表当前素数数量
      if ((long long)i * i <= n)
        for (int j = i * i; j <= n; j += i)
          // 因为从 2 到 i - 1 的倍数我们之前筛过了，这里直接从 i
          // 的倍数开始，提高了运行速度
          is_prime[j] = 0;  // 是i的倍数的均不是素数
    }
  }
  return p;
}
```

优化：对于 整数n来说 $\sqrt{n}$ ~ n的合数来说他们的质因子一定是在 1~$\sqrt{n}$所以我们只需要求解 1~$\sqrt{n}$的质因数的个数，并且把$\sqrt{n}$ ~ n的合数都记录下来就知道1~n的质数个数了

```
// C++ Version
int n;
vector<char> is_prime(n + 1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i * i <= n; i++) {
  if (is_prime[i]) {
    for (int j = i * i; j <= n; j += i) is_prime[j] = false;
  }
}
```

线性筛

我们发现埃氏筛还是有很多重复的筛过的，例如 30 = 3 * 10 = 2 * 15 所以我们思考优化筛过的重复的通过x = $p_1^{a_1}$ * $p_2^{a_2}$ * ..... * $p_n^{a_n}$ 我们发现我们每个数都有一个最小素因数，所以我们可以用这个思想，用一个数x去标记素数 * x，直到 x % 素数 == 0 直到x遇到他的最小素因数y 因为x也是由y标记的，因为我们再想用x去标记的话就是至少比 y大的质数，但是x包含y，所以违背了我们的初衷

```
// C++ Version
void init() {
  for (int i = 2; i < MAXN; ++i) {
    if (!vis[i]) {
```

```
      pri[cnt++] = i;
    }
    for (int j = 0; j < cnt; ++j) {
      if (1ll * i * pri[j] >= MAXN) break;
      vis[i * pri[j]] = 1;
      if (i % pri[j] == 0) {
        // i % pri[j] == 0
        // 换言之，i 之前被 pri[j] 筛过了
        // 由于 pri 里面质数是从小到大的，所以 i 乘上其他的质数的结果一定也是
        // pri[j] 的倍数 它们都被筛过了，就不需要再筛了，所以这里直接 break
        // 掉就好了
        break;
      }
    }
  }
}
```